Sixth Sense Software

Group 2

Project Start Date: 20/09/2021

Project End Date: 20/10/2021

WiseFlow Hand-In Date: 02/11/2021

Mickey M. Christoffersen - mick0841@stud.kea.dk

Barnaba Barcellona - barn0082@stud.kea.dk

Bogdan Moldovan - bogd0410@stud.kea.dk

Christian P. S. Sandgreen - chri73eq@stud.kea.dk

Kyle B. Dudley - kyle0026@stud.kea.dk

Benjamin Poulsen - benj074d@stud.kea.dk

Victor Christensen - vict3687@stud.kea.dk

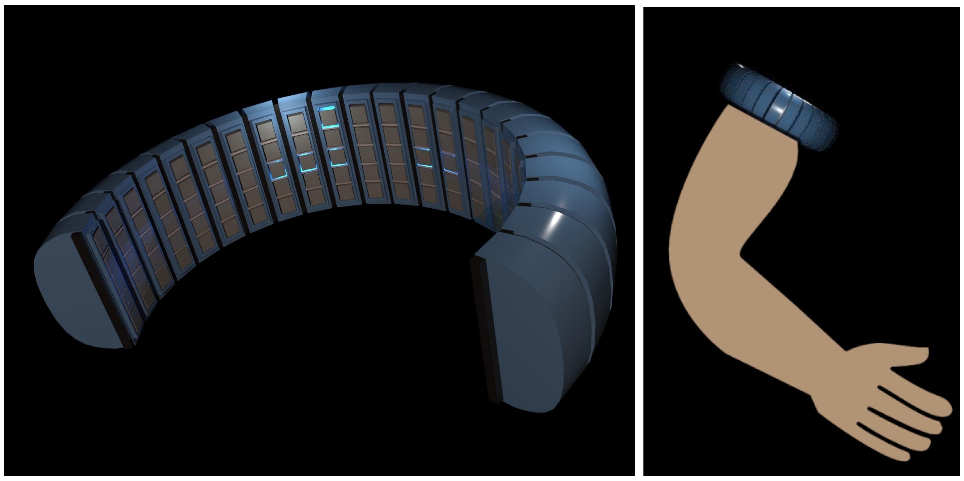
Link to GitHub: WAASE Team 2

Character & Word Count: (~ 30.720 & 5.430)

Table of Contents

1. Project Introduction

Sixth Sense is a Wearable Haptic Display Fig.1, in other words, a screen that you can feel. The wearable device aims to be an armband that connects to your smartphone through a Bluetooth module, allowing the user to "feel" the information sent by the last device. The type of data the device will be able to output might vary from a simple customized haptic ringtone, that allows the user to know who is calling his phone without directly looking at the screen or emitting any sound, to more complex functions such as text reading, watch, GPS compass, etc. The initial release of the device will feature an array of haptic engines (vibrating motors) that will activate in specific sequences sending a unique output around the arm of the user for each set of data that need to be displayed.



(Figure 1 - Mock-up)

1.1 Software Development Framework

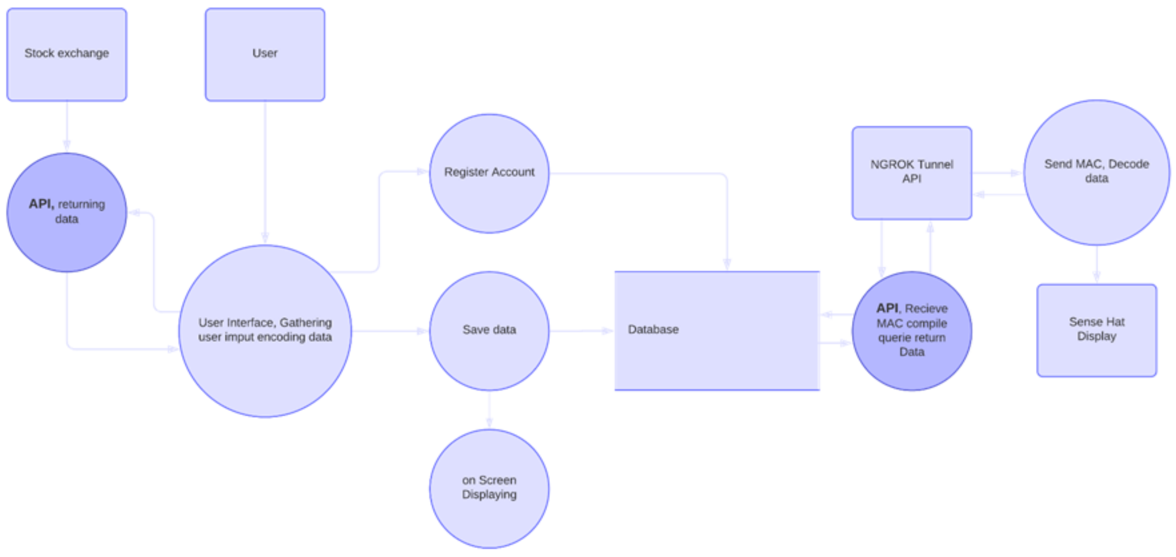
To plan out and organise the project we will be using a mixture of Scrum and XP. Scrum will be laying the groundwork for the planning of the whole project. This will be done through 4 sprints where the work of the sprints will be determined by the project backlog. We will be using user stories to populate the backlog and all user stories will be cleared with the product owner before being worked on[1]. Whereas with XP then we will only be using parts of it like pair programming and refactoring as examples[2].

[1] "What Is Scrum?" *Scrum.org*, https://www.scrum.org/resources/what-is-scrum.
[2] "What Is Extreme Programming (XP)?" *Agile Alliance* |, 10 Mar. 2021, https://www.agilealliance.org/glossary/xp/.

1.2 Description of the System

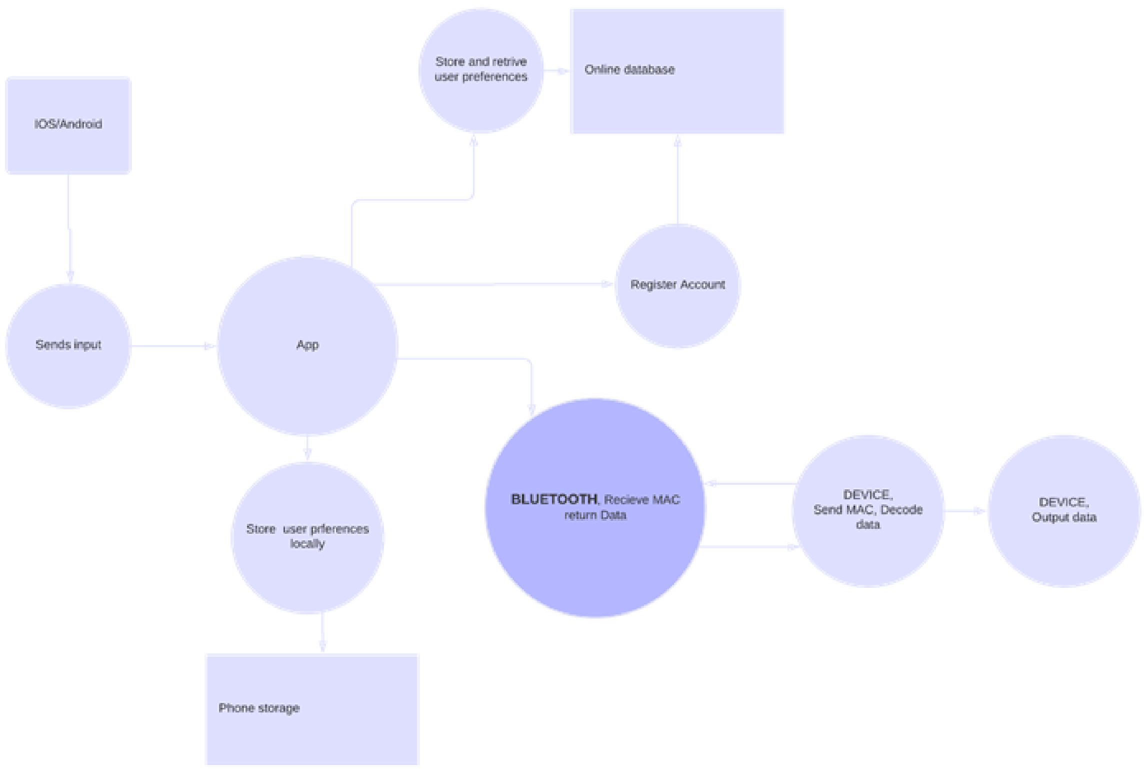
The system will consist of a python GUI on a local computer that handles the user inputs. The GUI will be able to allow the user to login to their account and create a new account. After login in the user can register a new device to their account or use a device that has been registered at an earlier point. After this stage the user can send either a stock price, direction or a clock command to the database through a MySQL connector to the database. The registered Raspberry Pi can then retrieve the command from the database through an API. The API used is a local hosted flask API that gets tunneled from through ngrok to a global domain that the Raspberry Pi can connect to. The MySQL database is hosted in the cloud on Microsoft Azure to make it accessible at all times instead of having a local computer being the server and thereby having to be constantly running for the whole project.

1.2.1 Current System



(Figure 2 - DFD Diagram (Current System))

1.2.2 Final Product System


Store and retrive user preferences
Online database
IOS/Android
Register Account
Sends input
App
BLUETOOTH, Recieve MAC return Data
DEVICE, Send MAC, Decode data
DEVICE, Output data
Store user prferences locally
Phone storage

(Figure 3 - DFD Diagram (Current System))

1.3 Description of the technologies

The scope of this project is to develop a Minimal Viable Product and IT system of the software (described by the DFD in figure 2) for the aforementioned device (wich final release of the system is described by figure 3), the development team aims to create a proof of concept system that allows the user to send inputs through a GUI on a pc to a Raspberry Pi device. The system will use a Microsoft Azure hosted MySQL database to temporarily store the inputs. The Raspberry Pi is then accessing the data inputs on Azure through an API and ultimately displaying them on the Raspberry Sense Hat. The GUI will further be replaced by an Android/IOS app as well as the online MySQL database by a MySQL lite local database on the smartphone, and the raspberry by a custom made PCB, therefore the final release of this project is just a proof of concept for Sixth Sense.

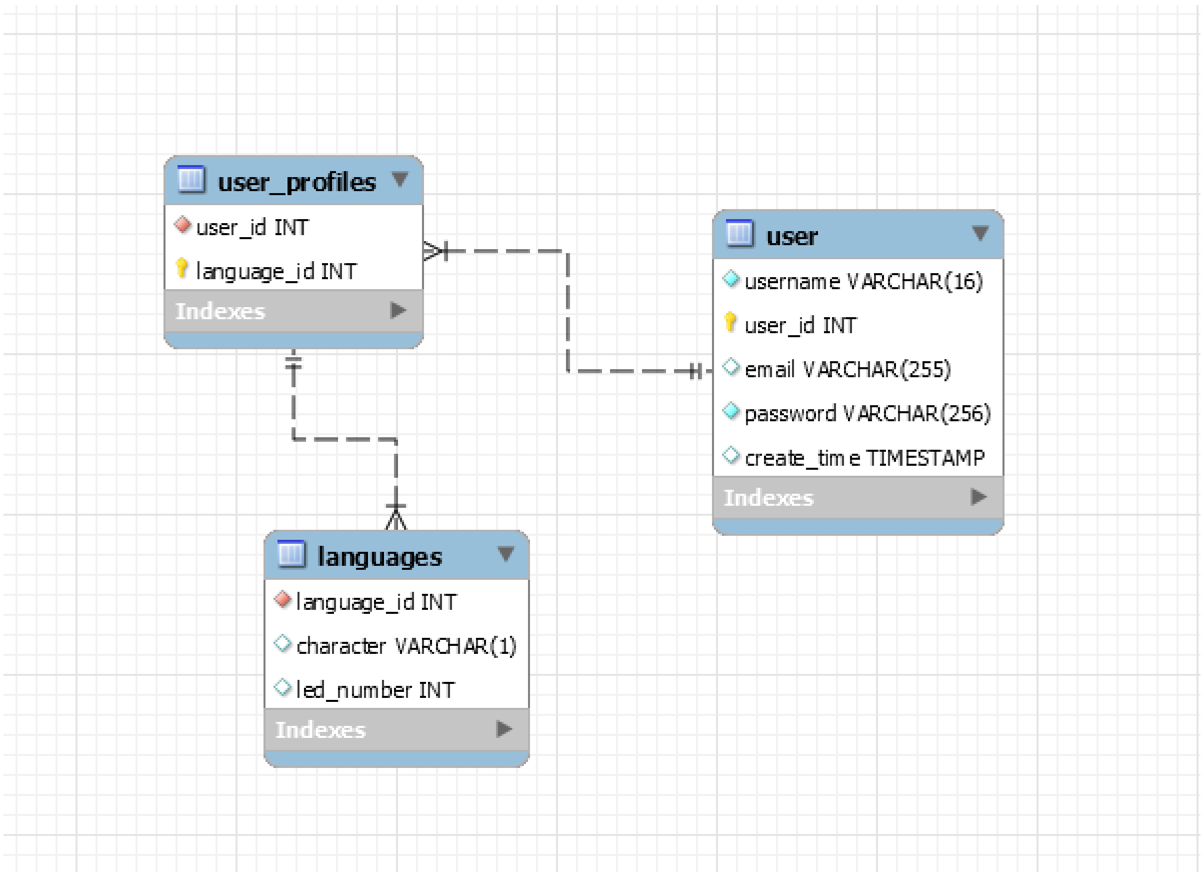
1.3.1 Python 3

The project is entirely developed in Python 3, as it is a very powerful programming language supported by a very large community of developers and extensive tutorial and resources available on the web.
Among other reasons python was our language of choice due to the fact that everybody in the group had previous experience with it and had knowledge about which modules could be used for the project.

1.3.2 Microsoft Azure & MySQL

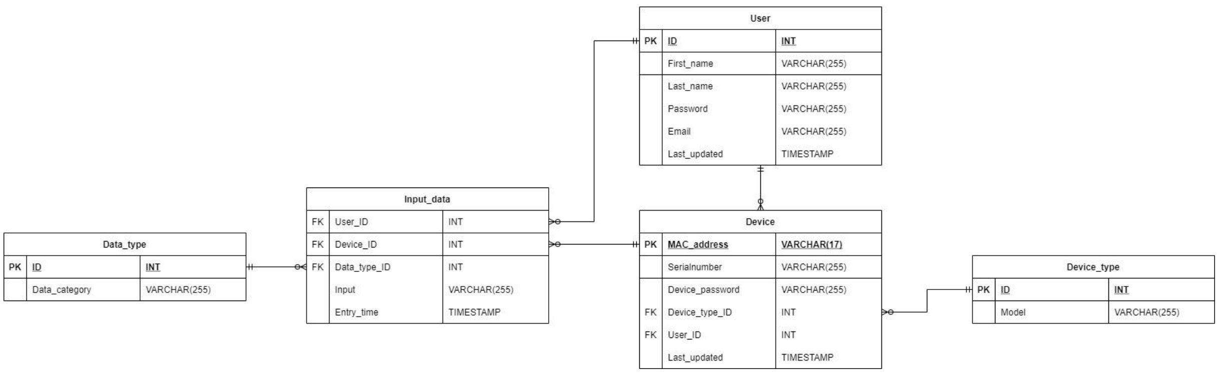
For this project we will require a database and for it we will use MySQL and Microsoft Azure. MySQL was chosen because it can be easily connected to Python and Azure allows all group-members to collaborate simultaneously through a cloud-based database.

Below is the first preliminary version of the database visualized as an ERD diagram.



(Figure 4 - ERD Diagram (Version 1.0))

After the first preliminary version was tested on Azure it was updated to 2.0. The new version of the database was updated to handle the needed data for the project. This meant that there was a need for new tables that could both handle information about the device and the data for the device. To secure a proper normalisation of the database the data_type and device_type tables were also added to reduce potential data redundancy.



(Figure 5 - Relational Database Model (RDBMS) (Final Version))

1.3.3 Ngrok

Ngrok allows us to facilitate and unveil the API running on our localhost making it available publicly with a generated URL on the ngrok.io domain while exposing our local web server to the NET. We create a HTTP tunnel by using the PORT number from our localhost or port desired in code f.x (PORT=5001). This allows us to tunnel requests from the NET to our local machine when it's behind a secure wifi network or firewall, passing network barriers like port blocking routers. We used the web framework Flask to create our API using Python language. Our Flask application runs on the localhost we mention throughout this document.

Downside to Ngrok is that it depends on our Flask server. If the server stops, the tunnel for the API shuts down and data is not available. As opposed to having an admin interface for our portal which Contoso would have provided. Moreover the given HTTP address is not static and will need to be updated each time the tunneling process is being restarted.

1.3.4 Tkinter GUI

The Tkinter module for python has been adopted to create a graphical interface, the module provides a Python native framework simple to use and easy to integrate with the rest of the system which share the same programming language.
Tkinter allows users to generate pop-up windows containing buttons and images, therefore to interact with the system without the need of command prompt or console.

1.3.5 Hardware

The system features a Raspberry Pi 3 as a proof of concept for the final mobile wearable device, the reason why we are using Raspberry Pi is because it is a simple and portable tool to develop hardware projects on. Furthermore the Raspberry Pi has the advantage of being able to run python and thereby allowing us to use python for both the GUI and the Raspberry Pi. in conjunction with the Raspberry we have used a Sense Hat to display data on the device this has allowed us to avoid electro mechanical complications and rely on libraries facilitating the process of lightening the led display.

2. Project Backlog

The project backlog consists of a structured list of deliverables, prioritized to fit the sprints goal and overall scope of the project. For our project, we used ZenHub, an agile project management platform directly inside GitHub. The project backlog mainly encapsulates issues the 'user' wants to be able to do with our product. We used "Epic" labels which are *"a theme of work that contains several sub-tasks or Issues"*[3], which is seemingly the best way to group related issues together into a larger goals that spanned the entirety of our sprints[4].

2.1 Prioritized List of User Stories

The below listed User-stories are agreed upon with the product owner. This list is in a prioritized order:

1. As a user I want to be able to display data on the Raspberry Pi.
2. As a user I want to access the database through a UI.
3. As a user I want to be able to connect from the Raspberry Pi to the database through an API
4. As a user I want to be able to select a defined set of data to be displayed on the device.
5. As a user I want to show the current stock price of a given company of the Raspberry Pi
6. As a user I want a menu that allows me to choose between different devices after I login to the UI
7. As a user I want to connect the API to the database and extract specific data.
8. As a user I want to connect the Raspberry Pi to a locally hosted API.
9. As a user I want to be able to show a directional input on the Raspberry Pi's display.
10. As a user I want to be able to logout from the interface.
11. As a user I want the registration page on the UI to close itself after submitting.

[3] "Getting Started with Epics in ZenHub." *Getting Started with Epics in ZenHub*
[4] Ibid.

2.2 Estimate top 4-10 User Stories

Below are the top four user stories that we have used throughout the span of the project. These user stories were first negotiated with the product owner, as we believe these four to be of utmost importance for the user's experience and understanding of the product provided. The quality of the user stories that we estimated to be the four best ones down below followed the INVEST principle[5], which is a set of criteria widely accepted, as the building blocks of a quality user story.

As a user I want to display data on the Raspberry Pi.

This ensures that data can indeed be displayed on the Raspberry Pi device. This user story was broken down into tasks such as user inputs, directional keys display and current time display which are all tasks, which includes data, that the user is capable of performing and displaying on the Raspberry Pi. Value is thus provided for the user, although this is only the preliminary depth of the Raspberry Pi, however one still gets to understand and see the direction the development of the project is headed, albeit the end-product is still a bit far from reality.

As a user I want to access the database through a UI interface and store the user inputs.

For the user to be able to access the database through the UI interface, we had to have a connection from the python code to the database. For that we used the module named MYSQL.connector. MYSQL.connecter allowed us to store and update the column / tables in the database. So everytime the UI receives inputs from the user, it gets updated and then stored.
To make this work, we broke the User story down into 4 different tasks.

1. User interface
2. Login
3. Connect the code to the database
4. Store the data

[5] "What Does Invest Stand for?" *Agile Alliance* |, 26 July 2021

For the user to be able to store and save his data, we started to connect the database, and create an interface for the user. In the interface the user should be able to register and login so we could save the correct data to the correct user.
We estimated this whole process to take 18 hours in total.

As a user I want to be able to connect from the Raspberry Pi to the database through an API.

For the user to be able to connect to the database through an API on the Raspberry Pi, we had to find a way to host an API that it could connect to. The solution to that was to use a Flask API that was tunneled through Ngrok to a global domain that the Raspberry Pi could connect to.
The sub-tasks include;

1. As a developer I want to host the API on a non-local network.

Our API is visible across a network by relaying to Ngrok the port number which our local web server is listening for incoming requests on.

2. As a user I want to connect the API to the database and extract specific data.

We access our database by defining configuration variables using "app.config" and including database credentials user, host, password and db name. In the process configuration values are forwarded to the Flask object (app=Flask). The HTTP method "GET" is used to send a message, thereafter the server returns data. The MySQL cursor points at the MySQL connection, executes the MySQL command (fx. SELECT * FROM...) fetches and returns the data as a python list.

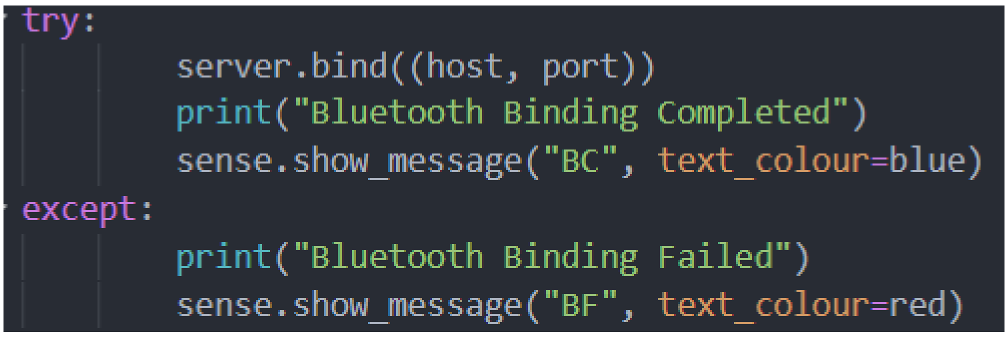
We estimated this whole user task process to take 17 hours in total.

As a user I want to be able to select a defined set of data to be displayed on the device.

For the user to be able to see the data on their device, we had to connect all our previous work. We had to connect the Raspberry Pi to the database and get it to show the data that is stored from the UI. To do this, we broke the user story down into 3 different tasks. We estimated the user story to take 13 hours.

Special Circumstances

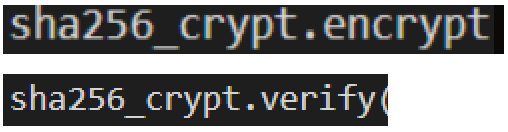
Try and except statements are used to catch and handle exceptions in the python code. Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside the except clause.



```
try:
        server.bind((host, port))
        print("Bluetooth Binding Completed")
        sense.show_message("BC", text_colour=blue)
except:
        print("Bluetooth Binding Failed")
        sense.show_message("BF", text_colour=red)
```

(Figure 6 - Try and Except statements (Snippet from python code))

Regarding security we chose to use password encryption, SHA-256 which is a cryptographic hash function that outputs a value that is 256 bits long.



```
sha256_crypt.encrypt
```

```
sha256_crypt.verify(
```

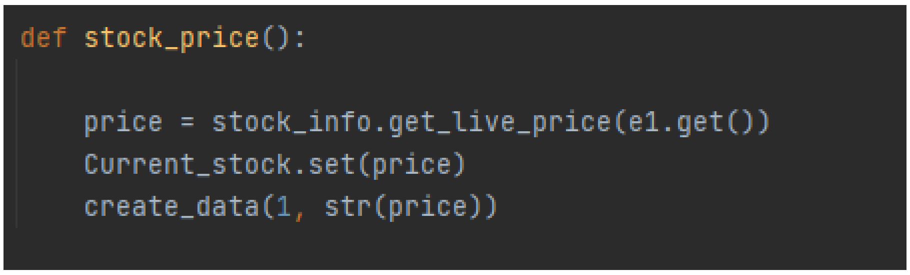
(Figure 7 - Password encryption and verification (Snippet from python code))

In the current version there are no user types in the database with special functions such as maintenance user, database user, or an admin. There is only one type of user that has access to everything.

Selected code examples

Underneath are selected code examples or the so-called *"gold nuggets"* that we believe are important to highlight and describe.

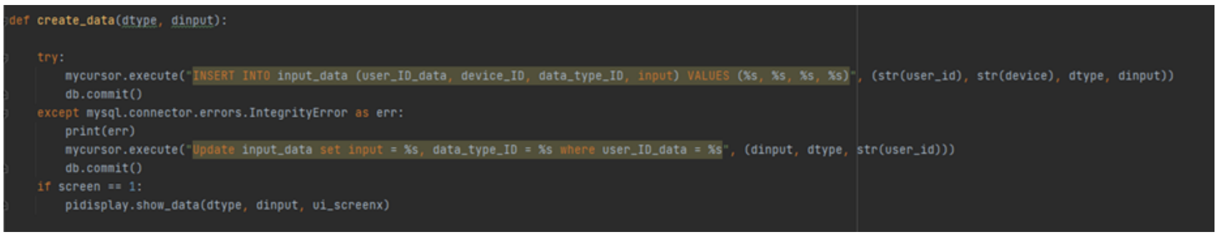
Figure 8 shows the function that retrieves the real-time price of a given stock. This function relies on stock_info.get_live_price a function embedded in the yahoo.fin module. The command returns a float and stores its value in the variable 'price', the .set() function will allow the current value to be displayed by the GUI label. The function will then run create_data(1, str(price))



```
def stock_price():

    price = stock_info.get_live_price(e1.get())
    Current_stock.set(price)
    create_data(1, str(price))
```

(Figure 8 - Stock_Price())

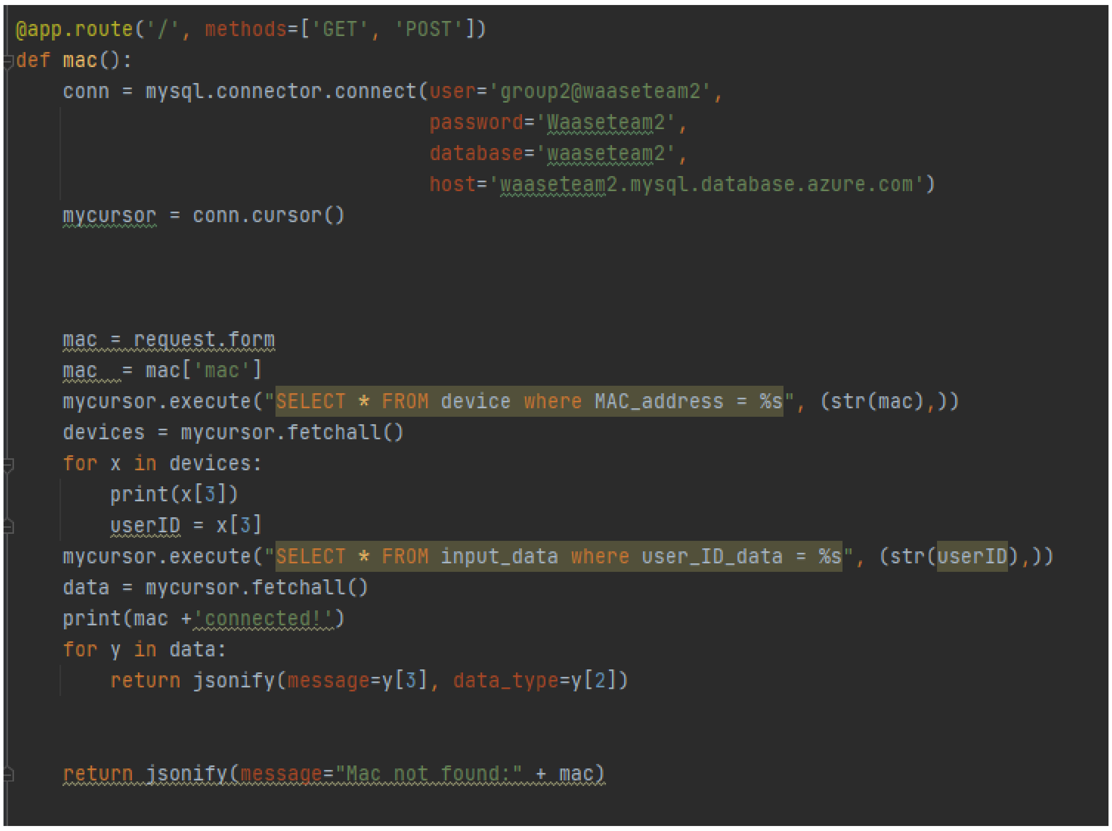
Create_data(x,y) fig. 9 through this function the newly created data set will be stored in the database, the function requires two arguments as data type (an integer value that define the type of data and indicated to the device how to display it) and data input, this last variable contains the string to output through the device. The function will at firsts attempt to create a new voice on the input_data table relative to the MAC address in use by the selected device, in case the mac address is already present in the table the try statement will return an exception captured by MySQL.connector.errors.IntegrityError and update the selected line instead of creating a new one, this process allow to minimize the storage consumption by storing only one data input per recorded device at each time.



```
def create_data(dtype, dinput):

    try:
        mycursor.execute("INSERT INTO input_data (user_ID_data, device_ID, data_type_ID, input) VALUES (%s, %s, %s, %s)", (str(user_id), str(device), dtype, dinput))
        db.commit()
    except mysql.connector.errors.IntegrityError as err:
        print(err)
        mycursor.execute("Update input_data set input = %s, data_type_ID = %s where user_ID_data = %s", (dinput, dtype, str(user_id)))
        db.commit()
    if screen == 1:
        pidisplay.show_data(dtype, dinput, ui_screenx)
```

(Figure 9 - Create_data())

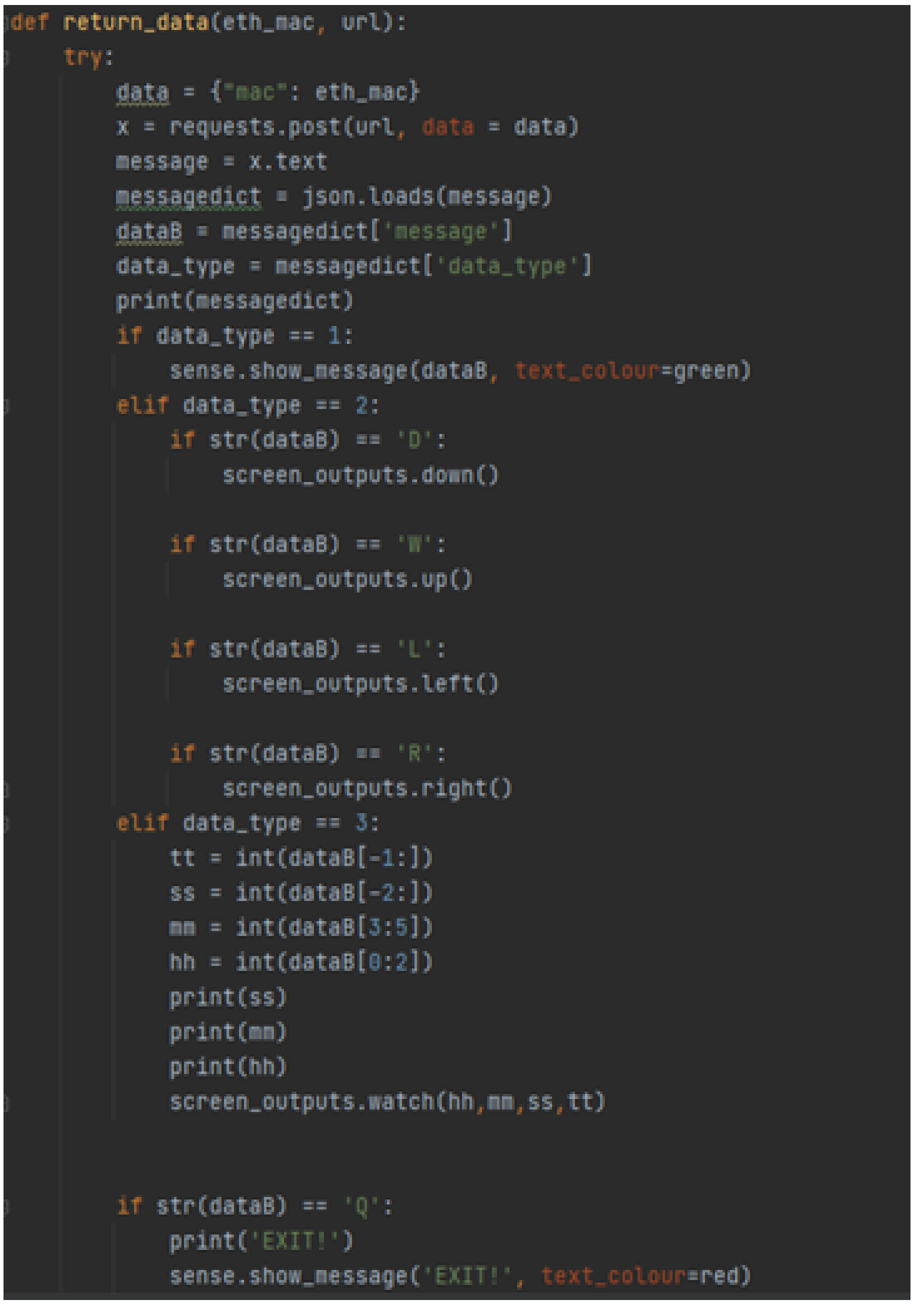
The flow of data will continue through the API fig. 10 where function mac() will capture the mac address sent by the Raspberry Pi through a POST request. The function will at first connect to the database and execute a query that will return the user_id('device owner') relative to the given MAC address from the 'device', this ID will be later used to identify the corresponding line of data stored in the input_data table through the GUI. The function will end with either returning a JSON scheme containing 'message' as the data to output on the device and 'data_type' as the type of data, or instead returning a JSON scheme containing an error 'message' whether the MAC address wasn't found within the 'device' table.



```
@app.route('/', methods=['GET', 'POST'])
def mac():
    conn = mysql.connector.connect(user='group2@waaseteam2',
                                   password='Waaseteam2',
                                   database='waaseteam2',
                                   host='waaseteam2.mysql.database.azure.com')
    mycursor = conn.cursor()


    mac = request.form
    mac_ = mac['mac']
    mycursor.execute("SELECT * FROM device where MAC_address = %s", (str(mac),))
    devices = mycursor.fetchall()
    for x in devices:
        print(x[3])
        userID = x[3]
    mycursor.execute("SELECT * FROM input_data where user_ID_data = %s", (str(userID),))
    data = mycursor.fetchall()
    print(mac +'connected!')
    for y in data:
        return jsonify(message=y[3], data_type=y[2])


    return jsonify(message="Mac not found:" + mac)
```

(Figure 10 - Flask API)

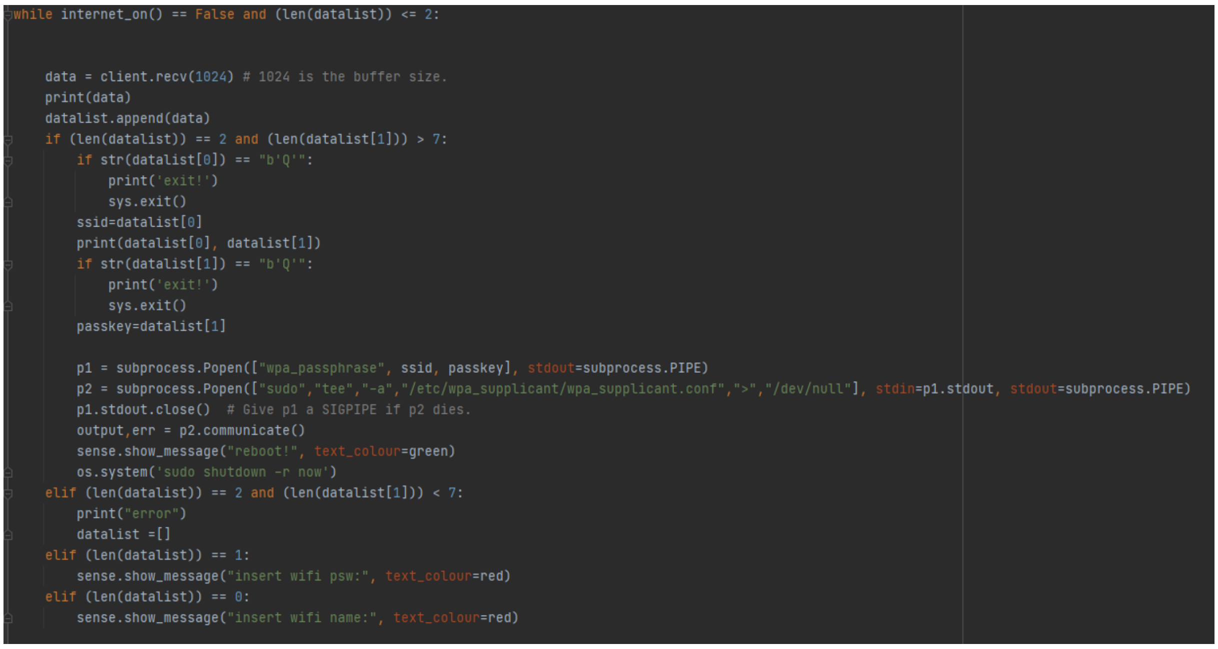
Finally, the data will reach the device where it is handled by function return_data (eth_mac, URL): fig. 11, this function takes two arguments, one containing the MAC address and the second containing the HTTP address where the API is hosted. Given the two arguments, the function will send a post request to the API and return a JSON object. The returned item is spilled into variable Data_type and DataB, a series of if statements that will initialize the displaying process addressing the right function accordingly with the type of data received.



```
def return_data(eth_mac, url):
    try:
        data = {"mac": eth_mac}
        x = requests.post(url, data = data)
        message = x.text
        messagedict = json.loads(message)
        dataB = messagedict['message']
        data_type = messagedict['data_type']
        print(messagedict)
        if data_type == 1:
            sense.show_message(dataB, text_colour=green)
        elif data_type == 2:
            if str(dataB) == 'D':
                screen_outputs.down()

            if str(dataB) == 'W':
                screen_outputs.up()

            if str(dataB) == 'L':
                screen_outputs.left()

            if str(dataB) == 'R':
                screen_outputs.right()
        elif data_type == 3:
            tt = int(dataB[-1:])
            ss = int(dataB[-2:])
            mm = int(dataB[3:5])
            hh = int(dataB[0:2])
            print(ss)
            print(mm)
            print(hh)
            screen_outputs.watch(hh,mm,ss,tt)


        if str(dataB) == 'Q':
            print('EXIT!')
            sense.show_message('EXIT!', text_colour=red)
```

(Figure 11 - Return_data())

The Raspberry Pi is set to run the given python script at boot fig. 12, this allows the application to be tested without the need to access the raspberry interface, in order for the system to be working under different networks the script will initially verify if the device is connected to the internet, if the device will result to be offline the script will request the user to insert a new Wi-Fi name and password through a Bluetooth connection, the given credential will be used to edit the wpa_supplicant.conf file. If the information is correctly added to the file the script will then reboot the device which should now be able to access the internet and execute the rest of the script.



```
while internet_on() == False and (len(datalist)) <= 2:


    data = client.recv(1024) # 1024 is the buffer size.
    print(data)
    datalist.append(data)
    if (len(datalist)) == 2 and (len(datalist[1])) > 7:
        if str(datalist[0]) == "b'Q'":
            print('exit!')
            sys.exit()
        ssid=datalist[0]
        print(datalist[0], datalist[1])
        if str(datalist[1]) == "b'Q'":
            print('exit!')
            sys.exit()
        passkey=datalist[1]

        p1 = subprocess.Popen(["wpa_passphrase", ssid, passkey], stdout=subprocess.PIPE)
        p2 = subprocess.Popen(["sudo","tee","-a","/etc/wpa_supplicant/wpa_supplicant.conf",">","/dev/null"], stdin=p1.stdout, stdout=subprocess.PIPE)
        p1.stdout.close()  # Give p1 a SIGPIPE if p2 dies.
        output,err = p2.communicate()
        sense.show_message("reboot!", text_colour=green)
        os.system('sudo shutdown -r now')
    elif (len(datalist)) == 2 and (len(datalist[1])) < 7:
        print("error")
        datalist =[]
    elif (len(datalist)) == 1:
        sense.show_message("insert wifi psw:", text_colour=red)
    elif (len(datalist)) == 0:
        sense.show_message("insert wifi name:", text_colour=red)
```

(Figure 12 - Wi-Fi connection/Bluetooth)

Status of implementation

The current state of the project is that the proof of concept is working as expected. The python GUI is working and is connected to the database. It allows the user to login, register a new user and register a new device. Furthermore it allows the user to display a stock price, direction and a clock both on the Raspberry Pi and on the computer's display. The Raspberry Pi has also successfully been connected to the database through an API. It was not done the way we first intended, which was to use Microsoft Azure's API. Instead it was done through a local hosted flask API that gets tunneled to a public domain by Ngrok. The Raspberry Pi is also connected to the WIFI which is setup through a smartphone that connects to the Raspberry Pi via bluetooth.

List of unfinished tasks from the scrum board

At the end of sprint 4 we still had some unfinished task that we weren't able to complete. The reason for this was due to the task being of a non critical function and therefore being down prioritized compared to other tasks. That combined with the fact that other tasks ended up taking longer than expected resulted in us running out of time and therefore had to leave them unfinished. The unfinished tasks are “As a user I want to be able to log out from the interface” and “As a user I want the registration page on the UI to close itself after submitting”.

Next steps of the project

- More user types such as database users that could analyze and plot data for visualisation. A maintenance user should be able to access everything related to the hardware in case of failure for troubleshooting and repairing. A developer user should have privileges to develop new features and new user inputs based on the future needs.
- Working hardware prototype that is usable, wearable and safe to use. The main components should be a battery, a charging port and the engines necessary for transmitting the input and a case to encapsulate everything.
- Expand the database for data collection that will be used to identify what characters/letters the users are struggling to understand. The reason for this is to make the language easier to learn for new users.

- Create a training process for the user to be able to understand the haptic language. This should be like an introduction to the device where the user will see in real time each character on the virtual screen, either on the phone or computer and then he will feel on his hand that character. After going through each character he will begin a testing phase structured like a quiz where he will try to understand different characters or words imputed by the device.
- Better looking and more intuitive interface that will make the user life easier and he will be focused on learning the device instead of trying to find his way around it.
- Mobile app that will connect to the device through bluetooth where the user could create an account and login to choose how the device will be used and what data should be imputed.
- A server running all the time to reduce the battery utilization of the device by computing and compiling the code in the database. The tasks that will be runned in the cloud should be carefully chosen and tested to figure out the energy consumption of both variants.

I. Project Process

The following subsection's goal is to emphasize and describe our work process during the project period. Over the course of the four sprints, we ran into many complications and problems, and it only got worse as the complexity of the product increased in the later sprints. This resulted in user stories etc., having to be altered or redefined to fit the new agenda. As with anything, this of course had to be renegotiated and settled with the product owner at one of the PO meetings. The project process overall was thus suffering in the early sprints, partly due to a lack of knowledge and partly because we had a product vision that was not quite feasible given the timeframe and complexity of said product, we learned this as we researched and got more knowledge on the subject. Therefore, the product release differed from what we had envisioned when we first started the project. Below is the work process broken down into each sprint, briefly describing what we worked on in a given sprint.

II. The work process

- What sprints there were and what user stories were worked on

Sprint 1 included researching, brainstorming and developing the product vision. Including the tools needed to bring the product vision to a real feasible product, in the end. We also created the start of the database in Azure, together with our first ER diagram.

In sprint 2 we decided to go with the python module “Tkinter” as our user interface, and started to create the fundamentals of the UI. We also decided which API we wanted for our Raspberry Pi, and finished designing our database.

In sprint 3 development of the product took off, first pieces of code were able to run on the raspberry, the first design of the language that we are displaying on the screens was created and we decided on the board length.

In sprint 4 we ran into some complications in regards to the API, as we thought we were able to host the API directly through Microsoft Azure, just like with the database. However that was not the case, and we had to come up with an alternative. The alternative resulted in the

API being hosted locally instead of in the cloud, which would have been the ultimate and most optimal solution. Another issue was showing some characters because they needed more space to be displayed on the board and we solved this issue by redesigning the haptic language.

Conclusively, the product vision started to take form into an end-product, able to deploy and run data. We had a stand-up meeting where we ran a demo of the entire process. From setting up the Raspberry Pi, to receiving and displaying data and user inputs. We had now successfully achieved our product vision's goal, and were rather satisfied with the result.

- An example of one of the PO meetings, what was planned on your part and how it went.

During the PO meeting for sprint 3 we discussed with the product owner the status of the project. One of the main things that was brought up during the meeting was the situation with Spike as our student account did not allow us to use it. Furthermore there was a discussion about the database and how we could improve the design a bit by changing the ID in the device tabel to use the MAC address from the Raspberry Pi. The implementation of daily standup meetings was also discussed as a way to improve the overall workflow for the next sprint.

- How you held your daily standup meetings

15-minute meetings discussing what to-do and what has been done from each group member, following a check-up of the tasks on our scrum-esque Zenhub board. In the early sprints we held most of our daily stand-up meetings online, however as things got more complicated we were more compelled to meet physically and do the stand-up meetings in person. This was mainly due to the increased complexity. However, we felt that the physical meetings were more efficient and effective, as we were able to better come to terms with eachothers in-depth knowledge and understanding of the end-products goal.

- When in held retrospectives

After the PO meeting in Sprint 3, we had a retrospective meeting which concluded the sprint. During the retrospective meeting we found that we should re-label our user stories and tasks to fit the new narrative. We should further update the database, using the MAC address as the unique identifier for each entry in the device tabel, and therefore a primary key within the database design. Moreover, the spike procedure was found to be productive, however, the examined technology resulted in a suboptimal outcome, due to lack of permission from Microsoft Azure in regards to the hosting of non-local API. Overall, the goal of the spike was to manage the API via API management in Azure. Azure/Contoso hosting our API would allow us to create, design, publish and deploy our API through a development portal in an API Development Environment on a centralized platform. For the sake of being efficient and organized this would have been the most optimal solution. After creating the Azure app, function app and API on Azure we made an attempt to deploy the API and publish it but were unable to do so due to subscription limitations and no access to the active directory. Lastly, we figured that some of our user stories were not very realistic, and were thus changed slightly. Some of them were too fixated on a non-important objective, which should just be a sub-task of a more objective and realistic user story. All in all, we realized that we had a bunch of work ahead of us in Sprint 4, and there were a lot of things that needed to be fixed or altered entirely. Conclusively, the retrospective meeting was successful and helped us identify commits to be made in the final sprint of the project.

III. The work process reflected

- Whether the scrummaster role worked, what problems you saw in it, and what you did to fix it.

Initially we started out with a scrum master but as the volume of the tasks increased and new tasks presented themselves, each team member contributed individually as the role of scrum master. The lack of a scrum master throughout all sprints can be attributed to the under-estimation of tasks and the workload associated with them. This is a scrum practice that should have been carried out over the entire project and proved effective when used.

- What were the main topics at your retrospective meetings?

Our main topics for our retrospective meetings were about what problems people had been experiencing since the last meeting. From there the focus of the meetings changed to how we could resolve those issues and thereby make sure that the project was able to keep progressing towards our vision for the project.

- If you had problems breaking down user stories into tasks.

We did in fact have problems breaking down user stories into tasks. Mainly because we were not really quite sure what a user story covered entirely. However as the project got more complex and more user stories became apparent, more tasks within the project backlog was also a necessity, which led to us figuring out how to break down user stories into tasks. Furthermore, redefining some user stories made the tasks less complex and more understandable.

- Whether you were spot-on with your estimates.

Our time estimates were off by a large margin. We underestimated the time it would take to research and figure out tasks.This happened due to a lack of knowledge upfront, lack of communication and lack of product transparency in the early sprints. All these things culminated into issues as the complexities of the end-product increased.

- How far into the process you found a rhythm that was productive.

We believe that mid-way through the process we started to find a rhythm that felt productive. We had more meetings, we communicated more, peer-programmed more, and overall worked more because we all were more or less on the same page and understood the processes and the direction we are moving towards.

- Other elements that have to do with trying to work in a scrum team.

Our team used Extreme Programming framework for software development because our project had dynamically changing requirements, risks caused by fixed time using new technology and the development team was small. From the five values of Xp we focused on feedback, courage, respect and simplicity while communications had its ups and down throughout the project. The most used practices by our team were pair programming, stories, weekly cycle, continuous integration and incremental design and spikes. We used incremental design to understand the perspective of the system and then we dived into details. For example when we display a character on the Raspberry Pi or virtual screen, we first tried showing one letter and from there we made then move to the left so that a long text will be readable on a 4 character display and then we add the rest. Pair programming was one of the most useful practices where two of us worked on a piece of code where one programmer knew how to use one module and the other specialized in conditional programming and structures and algorithms. The work was done online with the code going back and forth. Both programmers had their screen shared and we could watch the others' mistakes saving time for debugging. The reason we used pair programming is that we used it before and found out that it works very well. We also used spikes to test two different technologies and see which one works best in our case eg. if the computation should be done in the cloud or on the device hardware. Overall extreme programming was the best framework for us because it follows the natural instinct of doing things in our team.

Bibliography

"Getting Started with Epics in ZenHub." *Getting Started with Epics in ZenHub*, https://help.zenhub.com/support/solutions/articles/43000500733-getting-started-with-epics

"What Is Scrum?" *Scrum.org*, https://www.scrum.org/resources/what-is-scrum.

"What Is Extreme Programming (XP)?" *Agile Alliance* |, 10 Mar. 2021, https://www.agilealliance.org/glossary/xp/

"What Does Invest Stand for?" *Agile Alliance* |, 26 July 2021, https://www.agilealliance.org/glossary/invest/.